

Algorytmy optymalizacji inspirowane naturą – raport

Anna Czarnasiak – 268319, Marta Prucnal – 268368, Wojciech Bajurny – 268515

1 Wstęp

1.1 Cel projektu i opis problemu

Celem projektu było rozwiązywanie problemu optymalizacyjnego ogłoszonego przez platformę *Kaggle* w ramach konkursu *Santa 2025 – Christmas Tree Packing Challenge* [1]. Zadanie polegało na jak najgęstszym upakowaniu choinek w dwuwymiarowej przestrzeni. Każda choinka jest reprezentowana przez trzy parametry: współrzędne pozycji środka pnia (x, y) oraz kąt obrotu (deg).

Problem składa się z 200 niezależnych konfiguracji, gdzie dla każdego $n \in \{1, 2, \dots, 200\}$ należy umieścić n choinek w jak najmniejszym kwadratowym pudełku, minimalizując znormalizowaną powierzchnię ograniczającego kwadratu ($\frac{s_n^2}{n}$). Końcowa ocena stanowiła sumę wyników dla wszystkich 200 konfiguracji (Równanie 1).

$$\text{score} = \sum_{n=1}^N \frac{s_n^2}{n} \quad (1)$$

2 Wybrane metody i ich implementacja

Pełny kod projektu znajduje się w repozytorium pod adresem ¹.

Wybrano następujące metody:

- Algorytm zachłanny,
- Symulowane wyżarzanie (*Simulated Annealing* – SA),
- Ewolucja różnicowa (*Differential Evolution* – DE),
- Przeszukiwanie tabu (*Tabu Search* – TS),
- Optymalizacja rojem cząstek (*Particle swarm optimization* – PSO).

2.1 Algorytm zachłanny

Algorytm zachłanny jest samodzielnią heurystyką, która dostarcza rozwiązanie. Może być ono następnie wykorzystywane jako rozwiązanie początkowe dla metaheurystyk. Dodatkowo zawiera metody pozwalające przepakować choinki, ustalając ich nowe współrzędne.

Obiekt choinki (ksztalt) został zaimplementowany za pomocą biblioteki *GEOS*. Kształt choinki jest opisany przez 15 wierzchołków, które są skalowane względem współrzędnych środka (x, y).

2.1.1 Strategia pakowania

Algorytm zaczyna od umieszczenia pierwszej choinki w punkcie $(0,0)$ z kątem 0° . Następne choinki są dodawane do płaszczyzny iteracyjnie:

1. dla n -tej choinki tworzony jest obiekt, który umieszczany jest w punkcie $(0,0)$ z kątem 0° ,
2. następnie szukane jest najlepsze położenie – funkcja `findBestPosition`,
3. dla najlepszej znalezionej pozycji kolejno szukany jest najlepszy kąt obrotu – funkcja `findBestAngle`,
4. kolejno można „dopchać” wielokąt choinki w stronę środka, jeżeli zmniejsza to wielkość kwadratu – funkcje `tryMoveCloser`,

¹https://github.com/AnnCzar/AOIN_project.git

- na koniec danej iteracji obliczany jest bok kwadratu zwierającego wszystkie dotychczas umieszczone choinki – funkcja `calculateGlobalSquareSide`.

2.1.2 Znajdywanie najlepszego położenia

Odbywa się w funkcji `findBestPosition`, gdzie:

- rozważanych jest 45 kierunków – co 8° – wokół środka. Dla każdego kierunku (dx, dy) badane są kolejne położenia. Wielokąt jest przemieszczany w danym kierunku o `START_DISTANCE = 15,0` od punktu $(0,0)$.
- następnie choinka jest przesuwana z krokiem `STEP = 0,2` do środka, aż minimalny dystans między kolejną choinką nie przekroczy `MIN_DIST = 0,001`.
- W każdym kroku sprawdzana jest kolizja z już umieszczonymi wcześniej choinkami. Zostało to zoptymalizowane poprzez dzielenie przestrzeni na siatkę i sprawdzanie kolizji tylko w komórce, w której znajduje się choinka, oraz w ośmiu sąsiednich komórkach.
- Jeżeli nie wykryto kolizji, obliczany jest rozmiar kwadratu oraz odległość od środka układu. Wybierana jest taka pozycja (x,y) , przy której minimalizujemy najpierw bok kwadratu i ewentualnie odległość od środka układu.

2.1.3 Dobór kąta i lokalne „dopychanie”

Funkcja `findBestAngle` testuje kąty co 5° w $[0^\circ, 355^\circ]$ dla ustalonej pozycji. Dla każdego kąta:

- aktualizowane są wszystkie współrzędne,
- sprawdzane jest, czy występują kolizje,
- przeprowadzane jest „dopychanie” – `tryMoveCloser`,
- wybierany jest kąt minimalizujący kwadrat.

2.1.4 Obliczanie boku kwadratu

Za obliczenie boku kwadratu odpowiada funkcja `calculateGlobalSquareSide`, która wyznacza prostokątną obwiednię wszystkich choinek. Wybierane jest maksimum z szerokości i wysokości otrzymanego prostokąta.

2.1.5 Funkcje pomocnicze

- `packWithFixedAngles` – przepakowuje choinki dla zadanych kątów (dla SA/TS),
- `packWithFixedAnglesWindow` – przepakowuje choinki dla zadanych kątów (dla DE), obsługuje okno przesuwne,
- Spatial Hash – dzieli przestrzeń na komórki 15×15 , sprawdza 9 komórek (własna + sąsiednie).

Dla wszystkich metod metaheurystycznych skupiono się na optymalizacji jedynie kątów, co ograniczyło przestrzeń poszukiwań. W celu poprawy współrzędnych choinek wykorzystywano algorytm zachłanny dla kątów znalezionych przez metaheurystyki.

2.2 Symulowane wyżarzanie

Algorytm symulowanego wyżarzania zaimplementowano w celu optymalizacji kątów pakowanych choinek. Dodatkowo, co ustaloną liczbę iteracji, przekazywano zoptymalizowane kąty do funkcji `packTreesWithFixedAngles` w celu dodatkowej optymalizacji pozycji choinek. Częstotliwość przekazywania kątów do tej funkcji zmieniała się adaptacyjnie wraz ze wzrostem liczby drzew.

Parametry:

- Jako rozwiązanie początkowe przyjęto wyniki otrzymane przez algorytm zachłanny,
- temperatura początkowa $T_0 = 200$,
- współczynnik chłodzenia $\alpha = 0,98$,
- Sąsiedztwo – sprawdzanych było pięciu sąsiadów, gdzie jako operator generowania sąsiedztwa ustalono obrót o kąt wylosowany z rozkładu normalnego $N(0, \sigma)$. Wartość σ została zdefiniowana jako $\sigma = 5 + \frac{T}{T_0} \cdot 40$, dzięki czemu początkowa wartość wynosiła 45, a następnie wraz z kolejnymi iteracjami malała, co prowadziło do testowania coraz mniejszych obrotów,

- Częstotliwość przepakowywania choinek (*greedy-frequency*), określająca co ile iteracji przeprowadzane jest ponowne rozmieszczenie drzewek algorytmem zachłannym. Wartość ta malała wraz ze wzrostem liczby choinek:
 - 1 dla liczby choinek $n \leq 10$,
 - 25 dla liczby choinek $10 < n \leq 30$,
 - 50 dla liczby choinek $n > 30$.

2.2.1 Lokalne przeszukiwanie

Dodatkowo wzbogacono algorytm o lokalne przeszukiwanie wykonywane po zakończeniu głównej pętli optymalizacji. Polega ono na próbie obrotu każdego drzewka o kolejne kąty z wektora: [-5, -2, -1, 1, 2, 5] po zakończeniu wszystkich iteracji i sprawdzeniu, czy wynik po obrocie jest lepszy. W przypadku mniejszej liczby konfiguracji choinek niż 30 stosowano go razem z optymalizacją pozycji choinek, wykonywaną przez funkcję `packTreesWithFixedAngles`. Przy większej liczbie choinek nie zmieniano pozycji, ponieważ nie zaobserwowano satysfakcyjną poprawy względem czasu wykonania.

2.3 Algorytm Differential Evolution

Algorytm Differential Evolution został zaimplementowany w strategii *DE/best/1/bin*. Optymalizuje on wektory znormalizowanych kątów w oknie optymalizacji. W celu ograniczenia przestrzeni poszukiwań ustawiono precyzję kątów do 3 miejsc po przecinku (precyzja = $0,36^\circ$). Współrzędne (x_i, y_i) przepakowywane są dzięki funkcji `packWithFixedAnglesWindow`.

2.3.1 Parametry algorytmu

Parametry:

- Rozmiar populacji $NP = 30$,
- Maksymalna liczba iteracji $I_{max} = 100$,
- Współczynnik mutacji $F = 0,7$,
- Współczynnik krzyżowania $CR = 0,85$.

2.3.2 Inicjalizacja populacji z oknem

Populacja początkowa wykorzystuje rozwiązanie z algorytmu zachłannego. Dla każdej konfiguracji n :

- Kąty przed oknem ($j < window_start$) – pochodzą z poprzednich optymalizacji i nie są modyfikowane,
- Osobnik 0 – kąty z algorytmu zachłannego (ostatnie wartości z okna),
- Pierwsza połowa kątów w oknie – małe perturbacje wokół zoptymalizowanych kątów,
- Druga połowa kątów w oknie – duże perturbacje dla eksploracji.

2.3.3 Główna pętla ewolucyjna

Dla każdego osobnika x_i w każdej iteracji wykonywane są trzy operacje ograniczone wyłącznie do zmiennych w oknie:

1. **Mutacja** – generowany jest wektor mutant v_i zgodnie ze wzorem:

$$v_i = x_{best} + F \cdot (x_{r1} - x_{r2}) \quad (2)$$

gdzie $r1, r2$ to losowe indeksy różne od i .

2. **Krzyżowanie binarne** – tworzony jest osobnik próbny u_i , dziedziczący geny z mutanta v_i z prawdopodobieństwem CR
3. **Ewaluacja i selekcja** – wartość funkcji celu (fitness) obliczana jest jako bok kwadratu po upakowaniu choinek z kątami u_i . Jeśli $f(u_i) \leq f(x_i)$, osobnik potomny zastępuje rodzica.

2.3.4 Mechanizm okna optymalizacyjnego

Algorytm optymalizuje kąty tylko dla danych choinek w oknie, a resztę traktuje jako stałe. Dzięki temu ogranicza się liczbę zmiennych, co przyspiesza obliczenia, jednocześnie algorytm skupia się na drzewach, które najmocniej wpływają na aktualny rozmiar boku kwadratu. Wielkość okna zależy od liczby drzew n :

- do 5 drzew – optymalizacja wszystkich,
- 6-15 drzew – ostatnie 5,
- 16-50 drzew – ostatnie 8,
- 51-100 drzew – ostatnie 10,
- powyżej 100 drzew – ostatnie 15.

2.3.5 Naprawa rozwiązań

W celu uniknięcia utknięcia algorytmu, co 10 iteracji uruchamiana jest procedura przepakowania. Wybiera ona trzech osobników: najlepszego, najgorszego oraz losowego. Dla ich kątów wywoływana jest funkcja z algorytmu zachłannego, który próbuje znaleźć lepsze współrzędne (x, y) . Jeśli przepakowanie poprawi wynik, to współrzędne są aktualizowane dla danej konfiguracji.

2.4 Particle swarm optimization

W ramach projektu podjęto próbę przeprowadzenia eksperymentów dla algorytmu PSO. Zaimplementowano go w wersji podstawowej oraz rozszerzonej zgodnie z inspiracją z artykułu. [3] W wersji rozszerzonej cząstki podążają nie tylko w kierunku *personalBest* i *globalBest*, ale także *secondGlobalBest*. Ze względu na długi czas wykonywanych obliczeń nie udało się przeprowadzić miarodajnych eksperymentów. Jedna iteracja dla 10 drzew:

- wersja regularna, 50 cząstek – 300 s,
- wersja regularna, 100 cząstek – 562 s,
- wersja ulepszona, 50 cząstek – 275 s,
- wersja ulepszona, 100 cząstek – 521 s.

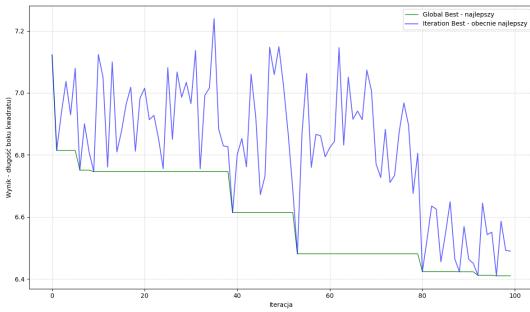
W Tabeli 1 przedstawiono wyniki 10 wywołań dla algorytmu PSO (30 cząstek) i ulepszonego PSO z parametrami

- $w_{max}, w_{min} = 0,9, 0,4$
- $c_1, c_2, c_3 = 1,49445, 1,49445, 1,9$

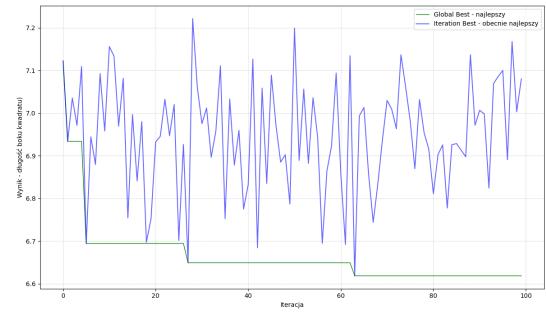
Tabela 1: Zestawienie wartości funkcji celu osiągniętych przez PSO i ulepszone PSO dla 10 drzew.

	Greedy (10)	PSOI (10)	PSO (10)
7,14		6,67	6,35
		6,48	6,54
		6,62	6,30
		6,40	6,32
		6,44	6,40
		6,60	6,28
		6,49	6,36
		6,58	6,37
		6,58	6,36
	ŚREDNIA	6,54	6,36
	SD	0,090	0,075

Rysunek 1 przedstawia przebieg najlepszej i aktualnej iteracji dla 10 choinek algorytmu PSO i ulepszonego PSO.



(a) Grupa 10 choinek



(b) Grupa 50 choinek

Rysunek 1: Wartość najlepsza i aktualna fitness w zależności od iteracji dla poszczególnych 10 choinek otrzymane przez pso i ulepszone pso.

Ulepszone PSO osiąga wyraźnie gorsze wyniki – możliwe, że wynika to ze zbyt dużego udziału *secondGlobalBest*, który zbyt mocno dominuje przy tak niskiej liczbie iteracji. Mimo to oba podejścia systematycznie zbiegają, jednak ze względu na długi czas obliczeń i niezadowalające wyniki zrezygnowano z dalszych eksperymentów dla tej metody.

2.5 Tabu search

W celu przystosowania algorytmu Tabu search do problemu dokonano kluczowych modyfikacji.

1. Dodano kryterium aspiracji – tj. sytuacja, w której dopuszczałyśmy złamanie listy Tabu. Zdecydowano się na powszechnie kryterium – jeśli w sąsiedztwie znajdziemy wynik, który znajduje się na liście Tabu, ale poprawiałyby globalnie najlepszy wynik – rozwiązywanie takie jest przyjmowane mimo zakazu.
2. Konieczność modyfikacji sąsiedztwa do przestrzeni ciąglej – zmodyfikowano pomysł z artykułu [2]. Tabu search blokuje identyczne rozwiązania, natomiast niemal niemożliwym jest, aby trafić na identyczne liczby zmiennoprzecinkowe. Z tego powodu wprowadzono promień – jeżeli różnica między kątami byłaby mniejsza niż zadany promień, to uznajemy te kąty za tożsame względem listy Tabu.

Parametry:

- Jako rozwiązanie początkowe przyjęto rozwiązanie losowe – w przypadku zachłannego algorytmu miał tendencje do utykania,
- Długość listy Tabu (T) – w zależności od liczby drzew:
 - $n = 10, T = 4$,
 - $n = 50, T = 8$,
 - $n = 100, T = 12$,
 - $n = 200, T = 20$.
- Granica dla poszczególnych kątów w Tabu $B = 1,8$, (różnica między danym kątem wynikowym a danym kątem z tabu) testowano też inne wartości – 3,6, 9, 18, jednak nie wpływały one wyraźnie na wyniki
- Sąsiedztwo – eksperymentowano ze zmianą kątów dla wszystkich drzew, a także dla części. Empirycznie pozostało przy zmianie kątów dla 10% drzew. Kąty generowano z rozkładu normalnego – $\sigma = 45$. Inne wartości σ nie wpływały istotnie na otrzymane wyniki.
- Liczba iteracji bez zmian, po których dokonywano przelosowania kątów $I = 20$

3 Eksperymenty i wyniki

3.1 Wstępne wyniki

W Tabeli 2 zestawiono wartości funkcji celu dla poszczególnych konfiguracji (10, 50 i 100 choinek) przed okresem wprowadzenia FFE.

- Wyniki dla SA przedstawiono dla wariantu sprawdzającego jednego sąsiada (nie dla 5, jak w ostatecznej wersji algorytmu) – $FFE \approx 1500$.
- Wyniki dla DE przedstawiono dla parametrów równych: $NP = 50$, $I_{max} = 300$, F oraz CR jest takie samo jak opisano w sekcji 2.3.1 – $FFE \approx 15000$.
- Wyniki dla TS przedstawiono dla parametrów – $FFE \approx 1500$.

Tabela 2: Zestawienie wartości funkcji celu osiągniętych przez badane metody dla poszczególnych konfiguracji.

Greedy (10)	SA (10)	DE(10)	TS(10)
7,14	5,71	5,92	6,42
	5,70	5,86	6,17
	5,71	5,90	6,22
	5,81	5,93	6,24
	5,69	6,09	6,40
	5,85	6,15	6,15
	5,76	6,07	6,08
	5,79	5,79	6,19
	5,63	5,63	6,24
	5,59	5,59	6,47
ŚREDNIA	5,73	5,89	6,26
SD	0,08	0,19	0,13
Greedy (50)	SA(50)	DE(50)	TS(50)
32,85	29,95	29,66	34,53
	30,02	29,92	33,61
	30,21	30,21	33,57
	30,01	30,20	33,20
	29,90	29,74	34,39
	30,09	30,44	33,92
	30,09	29,86	34,97
	30,01	29,73	34,59
	30,07	29,72	32,97
	30,08	29,86	34,15
ŚREDNIA	30,04	29,93	33,99
SD	0,09	0,26	0,65
Greedy (100)	SA (100)	DE(100)	TS(100)
63,56	60,82	60,30	64,73
	60,47	58,45	65,15
	60,69	58,79	65,99
	60,29	59,45	64,70
	60,39	58,54	65,62
	60,35	59,78	65,50
	60,44	59,10	63,71
	60,52	58,65	66,82
	60,11	58,85	66,68
	60,26	58,48	63,54
ŚREDNIA	60,43	59,04	65,24
SD	0,21	0,62	1,11

3.2 Porównanie metod

W celu porównania metod wykorzystano miarę, mówiącą o liczbie ewaluacji fitnessu (FFE, *Fitness Function Evaluation*), która dla pojedynczej konfiguracji ustalona została na 3000 dla każdego z algorytmów. Otrzymane wyniki przedstawiono w Tabeli 2.3.1. Ze względu na 5-krotne zmniejszenie FFE dla DE, zmieniono parametry w funkcjach przepakowujących: liczba kierunków = 90, START_DISTNACE = 10,0, STEP = 0,001 oraz PUSH_STEP = 0,01.

Tabela 3: Zestawienie wartości funkcji celu osiągniętych przez badane metody dla poszczególnych konfiguracji przy FFE=3000.

Greedy(10)	SA(10)	DE(10)	TS(10)
7,53	5,42	5,71	6,34
	5,53	5,69	6,29
	5,28	5,82	6,38
	5,34	5,72	6,29
	5,35	5,72	6,28
	5,35	5,52	5,96
	5,30	5,84	6,09
	5,25	5,88	6,20
	5,30	5,90	6,14
	5,30	5,82	6,20
ŚREDNIA	5,34	5,76	6,22
SD	0,08	0,11	0,13
Greedy (50)	SA(50)	DE(50)	TS(50)
34,01	29,57	28,63	31,52
	29,63	28,18	31,56
	29,86	27,19	31,39
	29,58	28,35	30,96
	29,55	28,08	31,14
	29,62	28,78	31,71
	29,52	28,10	31,43
	29,52	28,39	31,60
	29,62	27,97	31,45
	29,64	28,09	31,70
ŚREDNIA	29,61	28,18	31,45
SD	0,10	0,43	0,24
Greedy (100)	SA (100)	DE(100)	TS(100)
66,42	59,90	55,90	62,28
	59,86	56,56	61,50
	59,94	55,09	62,02
	60,04	56,28	61,89
	59,96	55,14	61,84
	60,07	55,22	61,64
	59,71	55,19	61,44
	59,80	55,65	61,05
	59,87	54,89	61,47
	60,03	54,50	61,72
ŚREDNIA	59,92	55,44	61,68
SD	0,11	0,64	0,35

W Tabeli 3 można zauważyć, że skuteczność poszczególnych metod silnie zależy od liczby konfiguracji. W przypadku dziesięciu konfiguracji choinek (od 1 do 10 choinek) najlepsze wyniki uzyskuje symulowane wyżarzanie, ewolucja różnicowa osiąga nieznacznie gorsze rezultaty, natomiast przeszukiwanie tabu wykazuje najwyższe wartości funkcji celu, choć wciąż są niższe niż wynik z algorytmu zachłannego. Wraz ze wzrostem liczby konfiguracji ewolucja różnicowa zaczyna przewyższać pozostałe metody, przy czym symulowane wyżarzanie pozostaje skuteczniejsze niż przeszukiwanie tabu. Takie wyniki można tłumaczyć między innymi rzadszym przepakowywaniem choinek w SA przy większej liczbie choinek.

W Tabeli 4 przedstawiono średnie wartości funkcji fitness uzyskane przez poszczególne metody dla dwustu konfiguracji choinek (grupy od 1 do 200 choinek) wraz z odchyleniami standardowymi.

Tabela 4: Średnie wartości funkcji celu dla poszczególnych metod wraz z ich odchyleniami standardowymi dla konfiguracji 200 choinek, przy czym w przypadku SA i DE wynik był uśredniany z dziesięciu powtórzeń, a dla TS – z trzech.

Liczba konfiguracji	Greedy	SA	DE	TS
	fitness	fitness		
200	131,82	120,42 ± 0,15	108,57 ± 0,81	123,02 ± 0,43

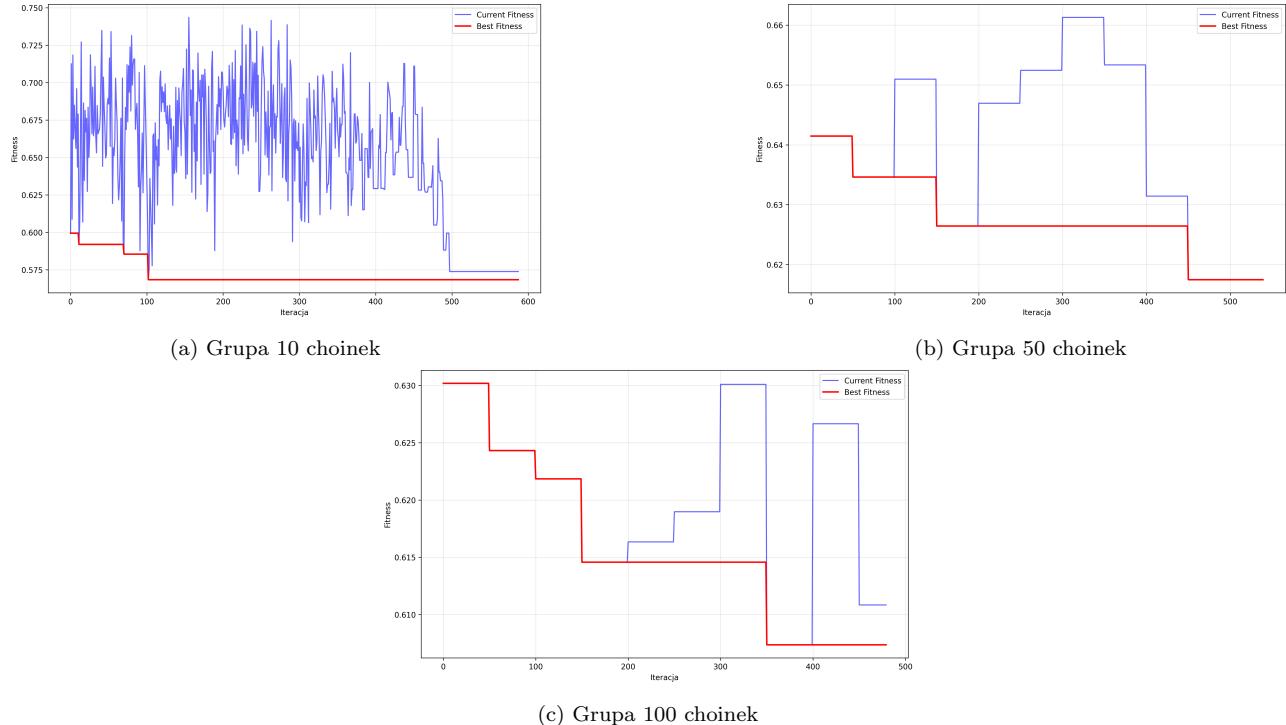
Tabela 4 potwierdza obserwację z Tabeli 3, że w przypadku większej liczby konfiguracji ewolucja różnicowa prowadzi do najniższych wartości funkcji fitness. DE osiąga średni wynik lepszy o około: 19,34% od algorytmu zachłannego, 12,48% od TS oraz o 10,35% od SA.

W Tabeli 5 przedstawiono średnie czasy trwania procesów optymalizacji dla poszczególnych metod. Należy zwrócić uwagę, że czasy te mierzono na różnych konfiguracjach sprzętowych, co ogranicza wiarygodność bezpośredniego porównania. Dodatkowo zauważalne są duże odchylenia standardowe w obrębie tej samej metody (czas mierzony na jednej konfiguracji sprzętowej), co może wynikać z nieizolowanego środowiska badawczego – w trakcie optymalizacji komputer był wykorzystywany do innych zadań, co mogło wpływać na dostępność zasobów systemowych. Mimo tych ograniczeń można zauważać znaczną przewagę czasową ewolucji różnicowej nad pozostałymi metodami. Wynika to z różnic w strukturze algorytmów oraz częstotliwości przepakowywania choinek algorytmem zachłannym.

Tabela 5: Zestawienie średnich czasów optymalizacji dla poszczególnych metod wraz z ich odchyleniami standardowymi.

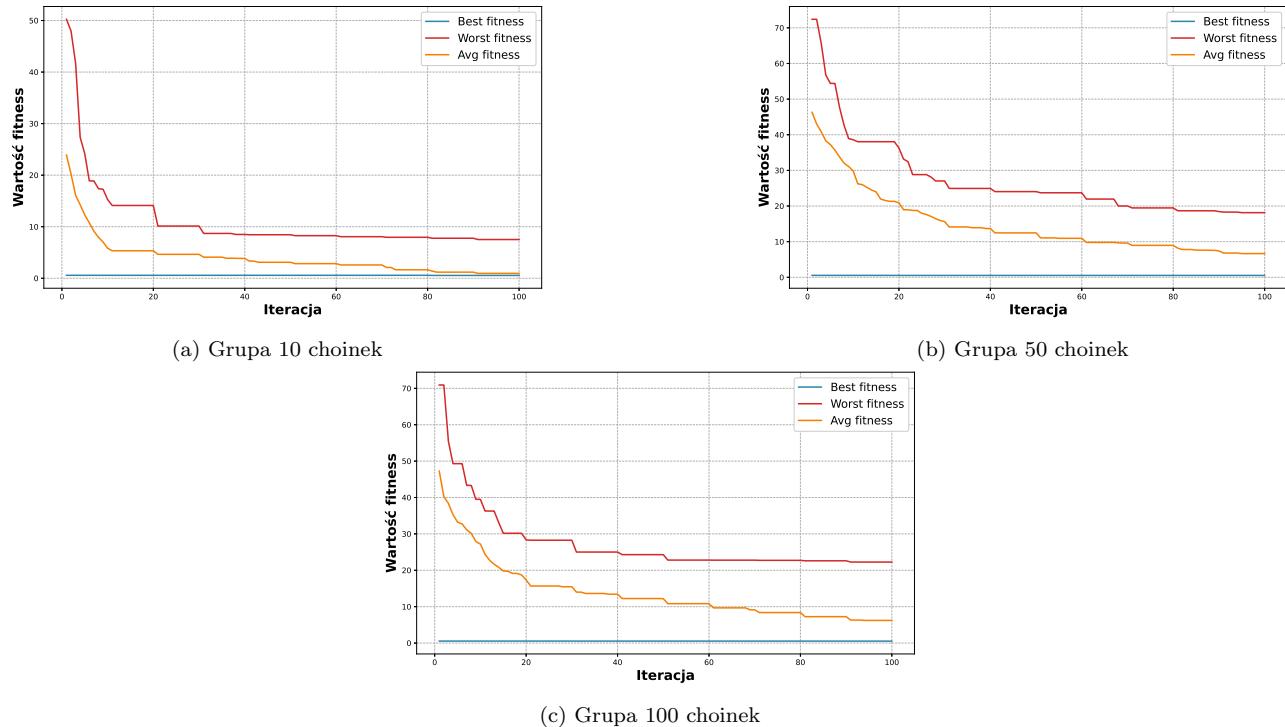
Liczba konfiguracji	SA	DE	TS
	\bar{t} (s)		
10	$1393,2 \pm 39,01$	$25,6 \pm 0,70$	$225,8 \pm 3,43$
50	$3946,4 \pm 282,68$	$579,8 \pm 21,95$	$1919,8 \pm 39,78$
100	$8814,9 \pm 397,07$	$2047,9 \pm 151,07$	$4767,1 \pm 16,15$
200	$20135 \pm 505,77$	$8977,7 \pm 923,43$	$13608,8 \pm 677,58$

Na rysunku 2 przedstawiono zależność wartości funkcji celu od numeru iteracji dla różnych liczebności grup choinek optymalizowanych za pomocą symulowanego wyżarzania. Można zaobserwować, że dla grupy 10 choinek proces przeszukiwania jest znacznie bardziej dynamiczny – przestrzeń rozwiązań jest intensywnie eksplorowana, co objawia się dużymi fluktuacjami wartości funkcji *current_fitness*, po czym algorytm stopniowo zbiera do optimum w okolicach 500. iteracji. Dla większych grup proces przeszukiwania jest wyraźnie mniej chaotyczny, a krzywa *current_fitness* charakteryzuje się mniejszą amplitudą wahań i bardziej płynnym przebiegiem. Takie zachowanie wynika z zastosowanej strategii przepakowywania – dla małych grup (do 10 choinek) choinki są przepakowywane w każdej iteracji, natomiast dla większych grup częstotliwość przepakowywania jest znacznie mniejsza (co 50 iteracji zarówno dla grupy 50, jak i 100 choinek). Prowadzi to do bardziej stabilnego, lecz wolniejszego procesu optymalizacji. Pomimo tego, we wszystkich przypadkach obserwuje się stopniowy spadek wartości *best_fitness*.



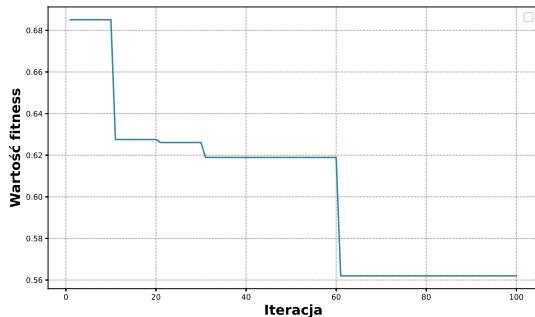
Rysunek 2: Wartość funkcji fitness w zależności od iteracji dla poszczególnych grup choinek otrzymane przez SA.

Rysunek 3 przedstawia przebieg najlepszej, średniej oraz najgorszej wartości funkcji celu w kolejnych iteracjach algorytmu DE dla 10, 50 oraz 100 choinek. Dla wszystkich trzech przypadków można zaobserwować szybki spadek wartości średniej i najgorszej w pierwszych iteracjach. Oznacza to, że populacja szybko zbiega do lepszych rozwiązań. Dodatkowo warto zauważyć, że krzywe średnia i najgorsza wartość znacznie przewyższają najlepszą, ponieważ do wartości fitness dodawana jest kara za każdą wykrytą kolizję. Wraz z kolejnymi iteracjami wartości te maleją, co wskazuje na naprawianie osobników i eliminowanie kolizji. Dla większej liczby choinek (50 i 100) zbieżność średniej i najgorszej wartości jest wolniejsza, jednak zachowany zostaje ten sam trend stopniowej poprawy całej populacji.

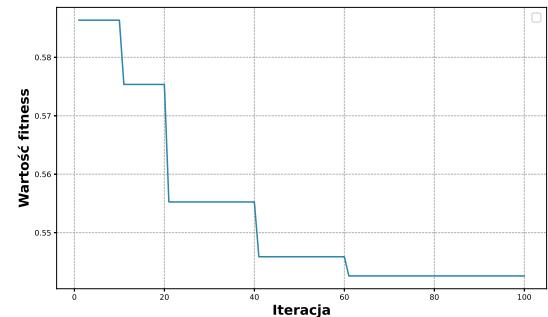


Rysunek 3: Wartość średnia, najlepsza i najgorsza funkcji fitness w zależności od iteracji dla poszczególnych grup choinek otrzymane przez DE.

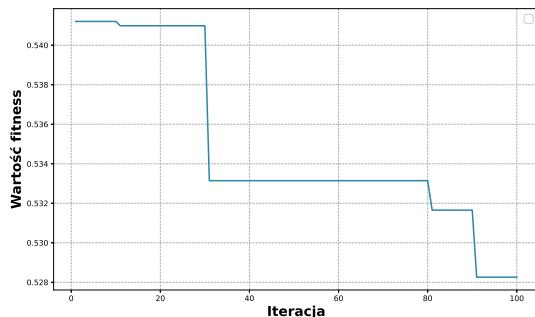
Rysunek 4 ukazuje zmiany najlepszej wartości funkcji celu w kolejnych iteracjach DE dla grup 10, 50 oraz 100 choinek. Wszystkie trzy wykresy mają charakter schodkowy, długie odcinki stałej wartości z nagłymi skokami w dół. Taki kształt krzywych potwierdza, że algorytm ewolucji różnicowej skutecznie wykorzystuje mutację i krzyżowanie do stopniowego ulepszania najlepszego osobnika w populacji.



(a) Grupa 10 choinek



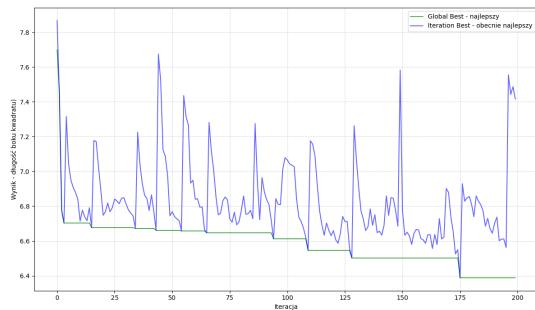
(b) Grupa 50 choinek



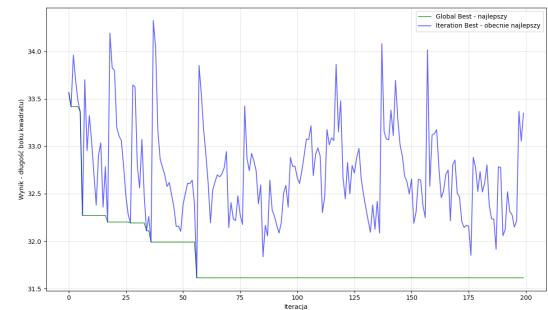
(c) Grupa 100 choinek

Rysunek 4: Wartość najlepsza funkcji fitness w zależności od iteracji dla poszczególnych grup choinek otrzymane przez DE.

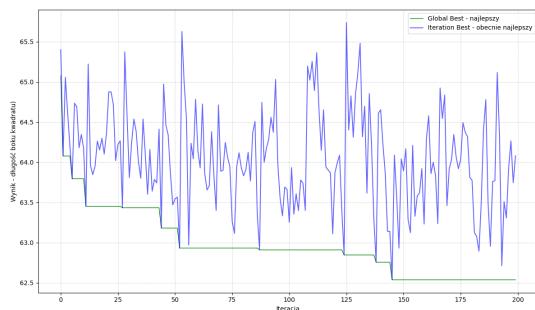
Rysunek 5 przedstawia przebieg najlepszej i aktualnej iteracji dla 10, 50, 100 oraz 200 choinek algorytmu Tabu Search. Na wykresach widać regularne piki – odpowiadają one przetasowywaniu populacji. Wykres dla 50 choinek w przedstawionym wywołaniu zbiegł przedwcześnie na drodze tego przelosowania, ale następnie nie zbliżył się do tego wyniku – należałoby zwiększyć I . Z drugiej strony dla 200 to przetasowanie pozwoliło wyrwać się z optimum około 130. iteracji. W przypadku 10 i 100 choinek wygląda to lepiej i algorytm eksploruje nowe przestrzenie i systematycznie poprawia rezultaty.



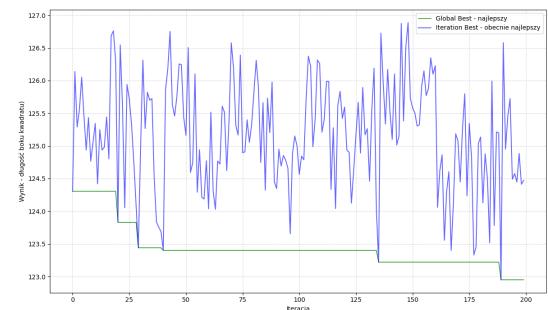
(a) Grupa 10 choinek



(b) Grupa 50 choinek



(c) Grupa 100 choinek



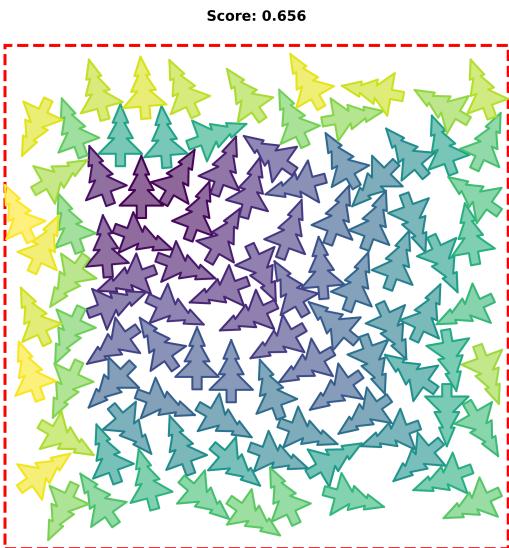
(d) Grupa 200 choinek

Rysunek 5: Wartość najlepsza i aktualna fitness w zależności od iteracji dla poszczególnych grup choinek otrzymane przez TS.

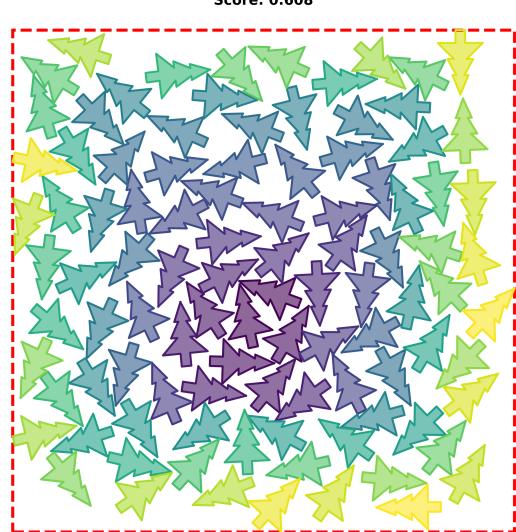
Na Rysunkach 6, 7, 8, przedstawiono przykładowe upakowanie choinek na podstawie uzyskanych zoptymalizowanych wyników z metod optymalizacji.



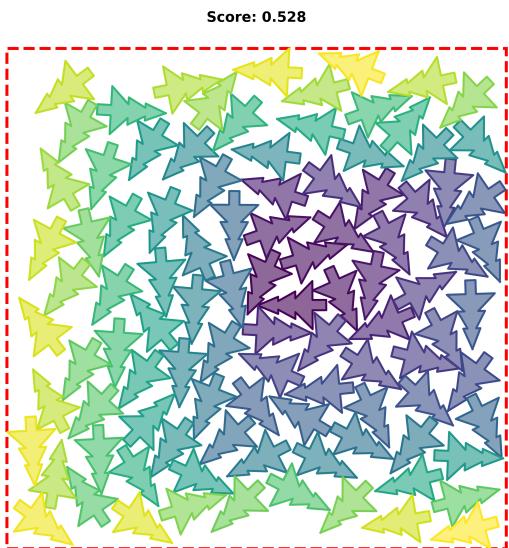
Rysunek 6: Wizualizacja spakowania 10 choinek przez metody optymalizacji.



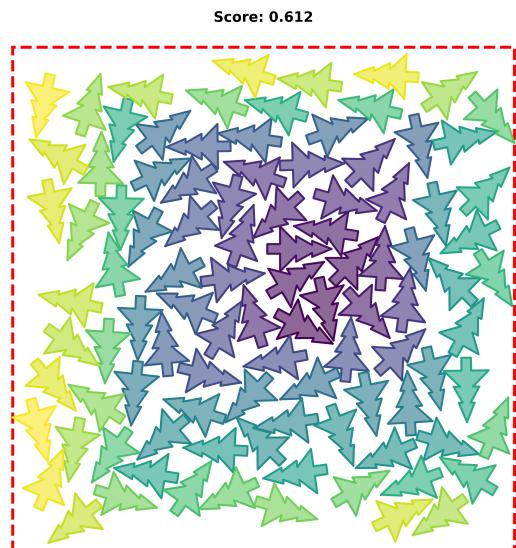
(a) Algorytm zachłanny



(b) Symulowane wyżarzanie

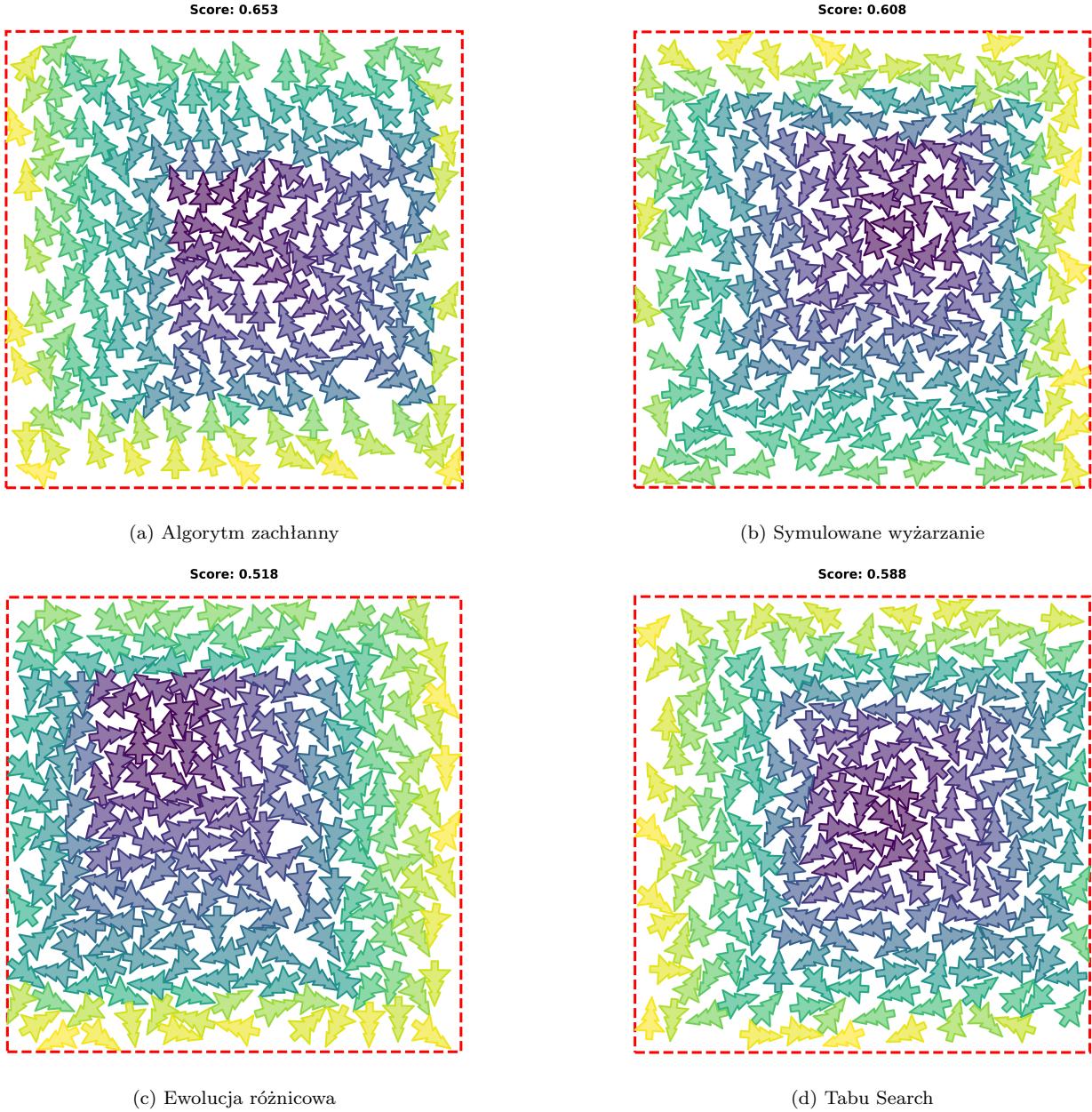


(c) Ewolucja różnicowa



(d) Tabu Search

Rysunek 7: Wizualizacja spakowania 100 choinek przez metody optymalizacji.



Rysunek 8: Wizualizacja spakowania 200 choinek przez metody optymalizacji.

4 Podsumowanie i wnioski

W niniejszym projekcie podjęto próbę rozwiązania problemu optymalizacyjnego polegającego na upakowaniu grup choinek (o liczności od 1 do 200) w możliwie najmniejszej przestrzeni dwuwymiarowej. W tym celu zastosowano różne metody optymalizacyjne: symulowane wyżarzanie, ewolucję różnicową, przeszukiwanie tabu oraz algorytm zachłanny, wykorzystywany zarówno jako samodzielna metoda, jak i procedura do przepakowywania rozwiązań.

Przeprowadzone eksperymenty wykazały, że wszystkie zaimplementowane metaheurystyki poprawiają rozwiązania uzyskane za pomocą algorytmu zachłanego. Przy tej samej liczbie ewaluacji funkcji celu ($FFE \approx 3000$) najlepsze wyniki dla małych konfiguracji (10 choinek) osiągnęło symulowane wyżarzanie, natomiast dla większych instancji (50 i 100 choinek) wyraźną przewagę uzyskała ewolucja różnicowa.

Pod względem kosztu obliczeniowego ewolucja różnicowa (DE) okazała się zdecydowanie najefektywniejszą metodą. Dla wszystkich testowanych konfiguracji czas jej działania był najkrótszy, przy jednoczesnym osiąganiu najlepszych wyników jakościowych. Efektywność ta wynika z zastosowania okna optymalizacyjnego. Symulowane wyżarzanie uzyskało wyniki konkurencyjne, a w przypadku 10 choinek nawet najlepsze, jednak wymagało wielokrotnie dłuższego czasu obliczeń. Przeszukiwanie tabu, pomimo zastosowania promienia do definiowania sąsiedztwa w przestrzeni kątów oraz kryterium aspiracji, okazało się najsłabszą metodą zarówno pod względem

jakości rozwiązań, jak i czasu trwania optymalizacji (choć krótszego niż w przypadku SA). Niższa efektywność TS może wynikać z faktu, że metoda ta nie jest najlepiej przystosowana do rozwiązywania problemów w przestrzeniach ciągłych, takich jak rozpatrywany problem.

Podsumowując, dla problemu minimalizacji rozmiaru kwadratu, w którym pakowane są choinki, najlepszym pojęciem jest zastosowanie ewolucji różnicowej w połączeniu z algorytmem zachłannym, szczególnie w przypadku dużych konfiguracji. Jeśli natomiast celem jest uzyskanie możliwie najlepszego wyniku dla małych instancji, zasadne może być wykorzystanie symulowanego wyżarzania. Należy jednak podkreślić, że średnia wartość funkcji celu jest wówczas jedynie o około 7,57% mniejsza niż dla DE, przy znacznie dłuższym czasie optymalizacji.

Literatura

- [1] Kaggle Santa 2025 - Christmas Tree Packing Challenge <https://www.kaggle.com/competitions/santa-2025/overview>.
- [2] Chelouah, Rachid, and Patrick Siarry. "Tabu search applied to global optimization." European journal of operational research 123.2 (2000): 256-270.
- [3] Shin, Young-Bin, and Eisuke Kita. "Solving two-dimensional packing problem using particle swarm optimization." Computer Assisted Methods in Engineering and Science 19.3 (2012): 241-255.